MECH 215

Lecture 3

Functions

Function Calls

- A function is a set of instructions that will be executed whenever the function is called.
- Each function is given a name, when this name is encountered in a program the function is called.
- When called the function performs the specified tasks and then ends.
- Once a function ends, the next line in the program is executed.

Function Calls

- In order to properly work, there are three parts of a function in the program.
  - The prototype: This defines whether the function returns a value (and what type of value) the function name and what input values are required.
  - The function call: This is a place in the main program where you wish to use the function.
  - The function code: This is where the actual C++ commands are placed to tell the function what to do.
- All three of these are required to make a function work in a program. Note that the function code may be located in another file.

Function Prototype

- The prototype statement is located close to compiler directives (such as #include <>).
- The statement says what type of value is returned:
- void, float, char, bool, int, double
- What is the name of the function
- What are the input values.
- void fun_out(int,float) is a function that returns no values, is called fun_out and has two input values one integer and the other floating point.

Function Call

- The function call is written in the main program and uses the name of the function and tells the program what values are input (specific locations in memory).
- fun_out(num_dollars,tax_rate). This function call invokes fun_out and copies the first input value the data in memory location num_dollars and the second input is from memory location tax_rate.
- The function will open new memory locations to work with, the original data is not touched.

Function Code

- The function code is the same as we have seen before with the following exception. In order to end the function, it is required that the line before the final bracket have return and the value that you want to return.
- For example return num_dollars*tax_rate would return the value equal to the product of num_dollars and tax rate to the main program and end the function.
- The function code can be written before or after the main program is you are not using call by reference. If you are using a prototype it must also include the same call by reference.

Memory

- The function will make a copy of the input values and store them in a different location in memory. Just because you use the same names in the program it does not mean that they are in the same location. To see the memory location of the data use & with the variable name.
- For example to print the memory location of the variable look_here you would write cout << &look_here;
- Often this is not an issue, but if you want the function to use the same memory location there is a bit of work.

Pass by reference

- Normally the program uses a pass by parameter, sending a copy of the value to the function.
- You can tell the program to use the same memory location (pass by reference) by writing in the function input variable with the & in the definition.
- Suppose we call the function how_now, which does not return a value to the main program, which has two input variable an integer num_of_cans and a floating point value num_of_liters. I want to use the call by reference in the nim_of_liters. This would take the form:
- void how_now(int num_of_cans, float& num_of_liters)

Calling functions

- Functions can call other functions and even call itself.
- When a function calls itself it is termed recursion.
- There is a risk that recursion can create an infinite series of calls (actually when the computer stack gets used up there will be an overflow and the program will end).
- How to use this without causing problems?

Recursion

- When a function calls itself there is a risk of creating an infinite loop. Typically the method to control this is to insure that within the function there is an end condition. An example is program 13 on the course website where each time the function calls itself the input is changed (in this case decreasing) and the function will terminate without calling itself when a certain input value is reached.

Recursion

- Recursion uses similar principles to looping within a program.
- Since the function calls itself, the input values are often used to control the ending of the recursion process.
- An example of this is program 13 where the factorial is computed using a recursive function call.

Recursion

- In this program the function fact_user calls itself unless the value1 is equal to 0, if this happens then the function will return 1 (ending the recursion process.
  - unsigned long int fact_user(int value1)
  - {
  - if (value1==0)
  - return 1;
  - // this is factorial of both 0 and 1
  - else
  - return value1*fact_user(value1-1);
  - }
- The function call within the function will open a new portion of memory.

Recursion

- In the function if the input value is equal to 1 then the value returned will be 1.
- If the input value is equal to 2 then the function will multiply the value 2 by the value returned from fact_user(value-1) which is fact_user(1) which is 1. This will lead to 2*1=2.
- If the input value is equal to 3 then the function will multiply 3 by the value returned from fact_user(2). Fact_user(2) will multiply 2 by the value from fact_user(1) this will lead to 3*2*1=6.

Programming

- Recursion will continue (opening new memory locations for each function call) until the function stops making function calls (in this case the return 1 line ends recursion) or the program counter (an internal register) runs out of space.
- Normally the same rules for looping will apply for recursion. Make sure that the stopping condition can be met, keep track of variables so as to not lose data.

Overloading Functions

- C++ has the feature that more than one function can have the same name. But have different parameter lists.
- If this occurs the compiler will look for the function that best meets the input parameter list and use that function.
- While this can make some programming easier, care should be taken as the compiler will select the function based on the following criteria: 1) Exact match, 2) Match using promotions (bool to int, char to int, float to double, etc…) 3) match using standard conversions (int to double).

Static Variables

- A static variable is initialed once when the function is called and then remembers the last value held before the function ends. Next time the function is called it will begin with the last value held.
- This is useful when counting the number of times a function is called as well as maintaining data between function calls.

Global Variables

- Global variables are defined near the compiler directives and are accessible for all functions.
- Note that if a new declaration of the same name as the global variable occurs, say in a function, the new declaration is treated as the local value (the global is hidden from the function). See program

Next Lecture

- In the next lecture we will discuss arrays.